X-Sieve: CMU Sieve 2.2
Subject: Comments to Hash Function Call
To: hash-function@nist.gov
Cc: Charanjit Jutla <csjutla@us.ibm.com>
X-Mailer: Lotus Notes Release 7.0 HF144 February 01, 2006
From: Charanjit Jutla <csjutla@us.ibm.com>
Date: Fri, 27 Apr 2007 20:13:55 -0400
X-MIMETrack: Serialize by Router on D01ML605/01/M/IBM(Build V80_M4_03042007|March 04, 2007) at
04/27/2007 20:13:56
X-Proofpoint-Virus-Version: vendor=fsecure engine=4.65.5502:2.3.11,1.2.37,4.0.164 definitions=2007-04-28_01:2007-04-27,2007-04-28,2007-04-27 signatures=0
X-PP-SpamDetails: rule=spampolicy2_notspam policy=spampolicy2 score=0 spamscore=0 ipscore=0 phishscore=0 adultscore=0 classifier=spam adjust=0 reason=mlx engine=3.1.0-0703060001 definitions=main-0704270164
X-PP-SpamScore: 0
X-NIST-MailScanner: Found to be clean
X-NIST-MailScanner-From: csjutla@us.ibm.com

Please find attached our extensive comments to the NIST call for hash funtions.
It starts with a two page executive summary.

Thanks,

Charanjit

(See attached file: nistcom.pdf) [below]

Evaluating a New Hash Function: Thoughts and Recommendations

R. Canetti, R. Gennaro, S. Halevi, C. Jutla, H. Krawczyk, T. Rabin

IBM T.J. Watson Research Center
Hawthorne, New York 10532

April 27, 2007

Abstract

This document is intended as a response to the call for comments by NIST related to the establishment of design and evaluation criteria for the upcoming hash competition. We start by presenting a list of specific recommendations for NIST's consideration and then follow with an article that expands on these recommendations and their rationale. We intend the list of recommendations also as an "executive summary" of the article for those not interested in the full details of our discussion.

Our approach is that due to the wide range of cryptographic applications for which the new hash function is intended (as implied by the FIPS 180-2 hash standard), NIST should select a relatively small set of core functionalities to serve as the basis for applicability of the new function, and derive from it a corresponding set of core security requirements that will serve as design and evaluation criteria in the competition. Moreover, we strongly recommend that NIST requires submitters of new functions to explicitly specify how to use the proposed function to achieve each one of the core functionalities, including the specification of how to key the function in keyed applications (for example, how to use the hash function to implement a PRF). We include detailed rationale for our recommendations as well as specific suggestions and considerations relevant to the planning of the upcoming hash competition.

[0]Contact author: Hugo Krawczyk, hugo@ee.technion.ac.il

.

Recommendations to NIST (executive summary)

The following is a list of recommendations regarding security criteria for the upcoming hash competition organized by NIST. Some of these recommendations are straightforward but some may require more elaboration. To this end, we attach to these recommendations an article containing a more detailed exposition of the relevant issues, including technical background and rationale, as well as further considerations and suggestions not included here.

Scope of applicability. In order to decide on relevant design and evaluation criteria for the new hash function, it is important that NIST explicitly states the scope of applicability of the new hash function. Lacking any specific guidance, we assume that this scope is to be derived from FIPS 180-2 (NIST's secure hash standard) which includes *all* uses of hash functions by Federal agencies, and hence includes virtually all major security protocols and standards. This, in turn, implies many diverse functionalities and a large variety of security requirements. An explicit statement from NIST confirming this scope, or correcting it, is essential for this process.

Core functionalities. Once the scope of applicability is clear, one can deduce a set of core functionalities that represent the main usages of hash functions in these applications. We recommend that NIST chooses such a set and uses it as a basis for providing guidance for the design and evaluation of the candidate hash functions. In particular, this set will serve as a basis from which to derive specific requirements for candidate functions. We recommend focusing on what we believe to be the four main applications of hash functions encountered in practice: digital signatures, pseudorandom functions (PRF), message authentication (MAC) and key derivation functions (KDF). Below we offer a set of specific security requirements derived from this list.

Modes of operation. Although the new hash function is to be used in all of the above applications, the exact form in which it will be used varies with each functionality. For example, a hash function can be used as a secure PRF when keyed in a certain way but can be totally insecure when keyed differently. Therefore, *NIST should require submissions to specify how to use the proposed function to achieve each one of the core functionalities*, including the specification of how to key the function. We refer to these different uses of the hash function as "modes of operation". In particular, while proposers of Merkle-Damgard functions, or slight variants of it, can use HMAC as a mode to provide PRF and MAC functionalities, those proposing new hash paradigms may need to come up with new ways for achieving these functionalities.

Note: The consideration of potentially new modes of operations to replace functionalities currently provided by HMAC raises some practical questions regarding the replaceability of HMAC in existing protocols and implementations where HMAC is currently "wired in" (see more discussion on this subject in the attached article)

Security requirements. After establishing the core functionalities intended for the new hash function one can derive a set of security requirements to be used as part of the design and evaluation criteria in the hash competition. The draft call by NIST lists preimage and collision resistance as well as "random-oracle indistinguishability" as the core security requirements. The first two are obviously required in the context of digital signatures and other applications. "Random-like behavior" is also needed in all the functionalities listed above. However, requiring a function to behave like a random oracle cannot be considered an *effective requirement*. In practice, one cannot build random oracles and, in particular, one cannot establish strict design and/or evaluation criteria to measure the "extent to which the algorithm output is indistinguishable from a random oracle" as stated in NIST's call. A more useful approach is to examine the new hash function under specific and well-defined randomness properties required by the above functionalities. We recommend that NIST decide on such properties and prioritize those requirements considered essential.

Here (and more extensively in the attached article) we discuss such a possible set of requirements. These include first and second preimage resistance as well as collision resistance (with variants of these notions depending on whether one treats these functions as keyed or unkeyed, what are the assumed preimage distributions with respect to which preimage resistance is required, etc.)

Requirements implied by the PRF functionality will depend significantly on the specific way (or mode of operation) proposed for use of the hash function as PRF. For example, in the case of Merkle-Damgard functions that implement a PRF family via HMAC, the essential property is that the keyed compression function be pseudorandom. For non-Merkle-Damgard functions the relevant properties to be argued will depend on the specific PRF mode suggested by the designers. Similar considerations apply for the MAC functionality except that distinguishing attacks are replaced with forgery attacks.

The key derivation functionality (KDF) is more complex as it is used in many different scenarios requiring a wide range of properties. The main requirement we identify is the ability of the function to preserve the entropy of the input distribution and "extract" this entropy into a close-to-uniform output. Due to the complexity of the arguments in this case we refer the reader to the attached article. There, in particular, we put forth the idea that candidate hash functions need to be analyzed with respect to the "universal hashing" property which we show to be of fundamental relevance to virtually *all* the considered functionalities.

Security thresholds and metrics. To make the preceding requirements more effective and well-defined, one needs to establish a set of criteria as for what is to be considered a successful attack against the above functionalities. Although for evaluation purposes one could say that an attack should perform better than a generic attack (i.e., exploiting some weakness in the specific design), many such attacks may be rather inefficient, and hence a burden on the design. In the attached article we address actual security thresholds for the main properties considered here and provide guidelines for the establishment of concrete criteria.

1 Introduction

This article is offered as part of the public comments requested by NIST in regards to the establishment of design and evaluation criteria to be used in NIST's competition for the selection of a new hash function to be added to the FIPS 180-2 suite. For convenience, we will refer to the new function expected to emerge from this competition as AHS ("advanced hash standard").

The first step towards deciding on security requirements is to establish the scope of application for the AHS. While NIST has not explicitly defined such a scope one can assume that it should be in line with the scope of the existing FIPS 180-2 standard. The latter states:

"This standard shall be implemented whenever a secure hash algorithm is required for Federal applications, including use by other cryptographic algorithms and protocols."

This seems to encompass a very large set of applications for hash functions, including widespread standards such as digital signature (and certification) standards, SSL/TLS, IPsec, other NIST-approved algorithms and protocols, and many more, all of which are in use by Federal agencies. Coming up with a hash function with such broad intended use is also in line, we believe, with what the industry at large would expect from a new standard hash function. It also implies a big challenge for designers of candidate AHS proposals, namely, the consideration of a large set of security requirements and properties for these functions as they arise from the many heterogeneous uses of hash functions in the above applications.

While an exhaustive enumeration of applications that use hash functions is hard to compile, one should identify the most widespread applications of these functions and derive from them a set of *core functional and security requirements* expected from the new function. In this article we will discuss at some length what functionalities and requirements should be included in such a set. The three main categories of applications we discuss are digital signatures, pseudorandom functions (and message authentication codes), and key derivation functions. While these are by no means the only uses of cryptographic hash functions they seem to encompass the most widespread applications and also imply requirements that are sufficient to ensure the security of other (though, not all) uses of these functions.

As we will see, the elements of collision (and preimage) resistance as well as randomness are encountered in all these applications. In this sense, the three security categories included in NIST's call – preimage resistance, collision resistance and "random oracle behavior" – are right on target. The latter property is particularly useful to cover virtually all uses of hash functions. Indeed, the "all-powerful" random oracle property (which, in particular, subsumes the two other requirements) may be enough to make the AHS usable in all existing applications. Unfortunately, this is only true in an abstract mathematical sense. In practice, one *cannot* build random oracles and, in particular, one cannot establish strict design and/or evaluation criteria to measure the "extent to which the algorithm output is indistinguishable

from a random oracle" as stated in NIST's call.

Therefore, it is important to be as careful, and as effective, as possible when determining a set of core design and evaluation criteria for AHS candidates. One contribution of this paper is in suggesting such a set. Moreover, we offer specific guidance towards establishing minimal security thresholds for a function to be considered acceptable (we note that such thresholds depend on specific parameters and types of attacks that vary from one functionality to another). In addition, we highlight central functional issues such as the distinction between keyed and unkeyed functions, and the need to consider families of hash functions rather than treating hashing as a single function. We also stress the importance of establishing *"modes of operations"* for the AHS, namely, well defined mechanisms that specify how to use the hash function to securely implement a given functionality (for example, a hash function may not directly implement a PRF but it may do so via the HMAC mode). We do not suggest that the competition should be extended to select standardized modes of operations, but rather that *submitters of new hash functions should be required to specify modes of operation to be used with their construction in a way that provides for the intended core functionalities.*

In particular, proposals that depart significantly from Merkle-Damgard will need to present (possibly new) modes of operations for replacing the functionalities commonly provided via HMAC. This, however, presents a "legacy" issue. Many existing protocols and standards (including SSL/TLS) have HMAC wired into their specifications and replacing HMAC with a new mode may be even harder than moving to a new hash function (e.g., replacing SHA-1 with a new function in these applications may be much easier than replacing HMAC with another mechanism).[1] In this case, NIST may need to consider whether to request, or at least establish as an advantage, "compatibility with HMAC". For example, a proposal that does not build on a compression function may still specify how to derive from its design a compression function usable, from the functional and security points of view, with HMAC.

2 Main Functionalities and Implied Requirements

Here we identify the main functionalities of hash functions as used in typical applications nowadays, and use this as a basis to distill a set of basic requirements (or desirable properties) for AHS candidates. We keep the presentation at an informal level and refer the reader to the literature for precise definitions of the notions discussed here. (In particular, see [15] for a formal treatment of several security notions for hash functions.)

[1] Even in the case of protocols that can work with non-HMAC schemes, such as IKE, it is likely to find HMAC as the only option supported in common implementations and hence hard to replace.

2.1 Digital Signatures

Digital signatures are the original application for which cryptographic hash functions were intended, namely, as short message digests for hash-then-sign type signatures such as standard RSA (PKCS 1) and DSS (FIPS 186). The main property implied by this use of hash functions is collision resistance since collisions in the hash function immediately translate into (chosen-message) forgeries against the signature scheme. This is also the property against which most of the recent attacks on hash functions have been directed and hence it is commonly accepted as the main requirement from the AHS.

However, it is essential to keep in mind that *collision resistance in itself is* **not sufficient** for the existing signature schemes to be secure. Indeed, there may be good collision resistant hash functions that when combined with algorithms such as DSA or RSA result in insecure signature schemes. The reason these examples are possible is that hash functions have another important role in these signature schemes, namely, countering the inherent weaknesses stemming from the algebraic structure (e.g., homomorphic properties) of the underlying signature algorithms. While the exact properties needed from the hash functions in these cases are not fully understood[2], one expects (or hopes) that a function with sufficient unstructured, or "random-like", behavior will not show exploitable weaknesses when composed with algebraic algorithms such as RSA and DSS. Here is where the requirements for "random-like" behavior of the hash function, that arise in the context of other applications, become relevant to the signature case. We discuss specific randomness properties of hash functions in the coming sections.

After having seen that full collision resistance is not sufficient for the security of signature schemes, it is also important to stress that collision resistance is also **not a necessary condition** for building secure signatures. We know, at least in theory, that weaker properties such as target collision resistance (TCR), also known as universal one-wayness, [13, 1] are sufficient for the construction of secure signature schemes. TCR hashing incorporates an element of randomization that makes off-line attacks, such as finding collisions in the underlying hash function, unapplicable to forging signatures. Fortunately, even though standard RSA and DSS signature schemes do not use TCR hashing, they can be adapted to enjoy the benefits of randomization and the reliance on weaker assumptions via simple message randomization. In particular, by doing so one obtains signature schemes that remain secure even after collisions on the hash function have been found and as long as the hash function enjoys some form of second preimage resistance [9]. We stress that given the perceived difficulty in designing a fully collision resistant hash function, it is important that proposers of new functions argue about the security of weaker properties as the above. An additional advantage of randomized hashing and TCR-type schemes is that their security is less affected by output truncation than in the case of collision resistance (there are no generic birthday attacks against TCR). In the setting of signature schemes, truncation of

[2] We note that for the standard deterministic version of RSA and for the standard DSS scheme, no proofs of security are known even when modeling the hash functions as random oracles (such proofs do exist for some randomized-hash variants of these schemes).

a hash output may be required, for example, when using a 512-bit hash function with the ECDSA scheme over a 256-bit elliptic curve.

Another basic property of hash functions relevant to the security of digital signatures is second preimage resistance (SPR). This notion, especially when there is an element of randomness in the choice of the first preimage, is a much weaker (hence harder to break) requirement from hash functions than collision resistance. In particular, SPR is required in order to ensure the long-term security of digital signatures since second preimage attacks can be used to change the value of a signed message even after a signing key (but not its corresponding public key) is no longer in use. In particular, such attacks do not resort to chosen messages. Fortunately, SPR is easier to achieve, in general, than collision resistance and must be required as a minimal requirement for AHS candidates.

When dealing with schemes that build on a basic building block such as a compression function, it is important to understand what properties of this building block are sufficient to ensure the collision resistance of the more general scheme. In the case of Merkle-Damgard we know that the lack of collisions in the compression function (called "pseudo collisions") is enough to ensure that the iterated hash function is collision resistant. In this case one can also study weaker properties of the compression function such as second preimage resistance or the recently introduced c-SPR property. (Interestingly, these two properties are violated by the MD5 compression function! [17]) While these properties are not sufficient to ensure the security of the composite scheme, they do serve as sanity checks for the whole design.

2.1.1 To key or not to key

In formal complexity-theoretic terms, collision resistance is not well defined for a single function (indeed, collisions can be "found" by a constant-time algorithm that has a colliding pair wired into it).[3] This is one reason why collision resistance should be considered as a property of a family of functions indexed by a key K. In this case, the required property is that for a random K it is hard to find collisions in the K-th function (note that K is given to the collision-search algorithm). There are, however, other, more concrete, reasons to develop a hash function as a family rather than a single function. For example, some applications salt[4] the function to prevent off-line and pre-computation attacks that target a specific function (UNIX password verification is such a case), or, in the case of KDF applications that use hashing for entropy extraction (Section 2.4), salting is necessary to enforce independence between the hash function and the input distribution. More generally, salting adds a dimension of randomization that helps against many attacks. The lack of well-defined salting usually leads to ad-hoc salting strategies such as using some of the bits

[3] A similar situation arises with preimage resistance. In this case, if one defines preimage resistance with respect to a specific distribution of inputs (e.g., the uniform distribution) then one can talk about a single function being preimage resistant. If, as often done, one requires this property for any input distribution, then one should talk about a family.

[4] We use the term "salt" instead of "key" for applications where the key is not secret.

of input as a key. This may or may not be the right thing to do depending on the function, and hence it is best that the designers specify how to do this keying/salting.

Having said this, we stress that many applications do require to define a single hash function, most prominently signature schemes (this, however, is not fully necessary if one resorts to the more secure TCR, or eTCR, based schemes). For these cases, designers of a new hash family should specify a specific salt value to be used in these situations. Also, we stress that a single salting strategy may not be optimal for all uses. For example, a Merkle-Damgard hash can be naturally salted via the IV. However, for building an eTCR family, as in [9], using the salt to xor it to input blocks is a better strategy. Moreover, a keying strategy that works well with non-secret keys may not be optimal for secret-key applications such as PRFs (we note that for secret-key applications we require the designers to explicitly specify how the function is keyed).

2.2 Pseudorandom Functions

The most common application of hash functions after signature schemes (and maybe even more widely used than digital signatures) is for building pseudorandom functions (PRF). This is a secret-key primitive (a family of keyed functions) whose outputs look fully random to a computationally-bounded attacker that does not know the secret key. Formally, the security of a PRF is stated in terms of *computational indistinguishability* from uniform outputs. Applications of PRFs include the construction of pseudorandom generators (e.g., using the PRF in "counter mode"), key derivation functions (expanding an initial key into multiple keys), key management (e.g., a centralized key management facility equips device i with secret key $K_i = PRF_K(i)$), instantiating MAC functions (see below), and many more.

Hash-based PRFs are common (they are often referred to as *"keyed hash functions"*) and used in innumerable standards including SSL, IPsec, SSH, etc. For this application, hash functions need to incorporate a secret key which is not necessarily part of the basic definition of the hash function. With Merkle-Damgard type of hash functions this is often implemented via the HMAC scheme (which serves as a keyed "mode of operation" for hash functions). In this case, the necessary and sufficient condition for the a hash function to provide a secure PRF under HMAC is that the underlying compression function (a well defined component of any Merkle-Damgard hash) be a fixed-length input PRF (keyed via its "chaining variable" input). In non-Merkle-Damgard designs, alternative keying methods, and their corresponding security properties, need to be specified and argued by the designers. In all cases, the ability to use the hash function for instantiating a PRF should be a central requirement for AHS. We note that alternative constructions of PRFs, e.g., based on block ciphers, exist, but there is a very large number of applications that use hash-based schemes (e.g., HMAC) to provide this functionality.

Additionally, we stress that the requirement that the hash function possess good pseudorandom properties (when combined with a secret key) can be seen as one of the best, and better defined, instantiations of "random-like properties" for hash functions, even though

being a PRF does not provide, in itself, any guarantee when using the same function(s) with a known key.[5]

2.3 Message Authentication Codes

Message authentication codes (MAC) are cryptographic functions used to authenticate information transmitted between two parties that share a common key. As in the case of PRFs, a MAC consists of a family of (secretly) keyed functions. The security requirement from MAC functions is *unforgeability*, namely, an attacker that does not have the value of a key, cannot compute a correct MAC value under that key on any message of its choice even after seeing other messages MACed using the same key. This is a strictly weaker requirement than PRFs and, indeed, PRF families also make secure MAC families but not viceversa.

One aspect in which MAC functions may be more demanding than PRFs is efficiency. In many applications PRFs are computed on a small number of inputs, for example as part of a key-exchange protocol, while MACs are computed on a large number of messages. This is one possible reason to have MAC functions that are not implemented via PRFs (it is also one aspect in which the security of a MAC may be more demanding since attacks using a large number of chosen or known messages may be more realistic against a MAC than against the PRF functionality). In today's practice MAC functions are often based on (keyed) hash functions, and the most common implementation is via HMAC (i.e., the same implementation as PRFs). Proposers of new hash function need to specify how to build MAC families using their design (also here, part of the specification is how to key the function) and on what basis can the security of such a scheme be argued.

2.4 Key derivation functions and entropy extractors

We have mentioned that PRFs, including hash-based ones, are often used for the derivation of multiple secret keys from an initial single key. This application assumes that one starts with a cryptographically strong key (say, a 160-bit close-to-uniform string) and uses it to key the PRF to derive further keys. In many cases, the initial "keying material" available to an application does not consist of a close-to-uniform (or pseudorandom) short string that can be used directly as a cryptographic key, but it rather comes in the form of a longer string drawn from a high-entropy source where the entropy is spread over the longer output (e.g. a 256-bit entropy source sampled from 1024-bit strings). In this case, one can hope that hashing the available keying material into a shorter string (shorter than the entropy)

[5]Note the apparent similarity between a PRF and a random oracle. Both cannot be distinguished from random functions. However, while random oracles (with efficient computation) cannot possibly exist, PRFs have efficient implementations (assuming existence of one-way functions or block ciphers). The crucial difference is the use of a secret key in the PRF application, something that does not exist in typical random oracle applications (e.g., in signature applications).

will produce a close-to-uniform output (in the above example, one would hash the sampled 1024 input into, say, 160 bits to obtain 160 close-to-uniform bits).

A hash function that can "extract" the entropy of a probabilistic source into a close-to-uniform output as above is called an *extractor* (we omit formal definitions; see [14, 7]). Cryptographic hash functions are commonly used as extractors in key derivation functions. Real-life examples include the practice to produce a cryptographic key by hashing a stream of bits generated via the sampling of system events (or user generated keystrokes) or even by hashing an (imperfect) random number generator. A different, and very common, example consists of the hashing of Diffie-Hellman values (exchanged in a DH protocol) down to a shorter string used as a shared cryptographic key.

This application of hash functions is so widespread and essential to the security of many cryptographic systems, that we believe it should be considered among the main functionalities to expect from the AHS. The question is how can one characterize the properties required for such a functionality. There is no single answer to this question since requirements vary with the details of the probabilistic source, its guaranteed minimal entropy and the number of bits required as output of the extractor. The simpler case is when the entropy of the source is significantly larger than the number of key bits that need to be generated. In this case, there are well-known families of functions (e.g., universal hash functions, see below) that can provably guarantee extraction of close-to-uniform bits for *any* such source.[6] In the case where the source and output have similar entropy one has to consider specific properties of the source or, for source-independent extraction, one has to resort to idealized assumptions such as the hash function behaving as a random function or a family of random functions.

Below, we will discuss the properties of universal hashing as a useful design and evaluation criterion for AHS with respect to several functionalities including entropy extraction. Finally, we mention that in the case of Merkle-Damgard hash functions it has been shown [7] that HMAC can preserve some of the extraction properties of its compression function. Submitters of new schemes should be encouraged (if not required) to argue about the extraction properties of their designs.

2.5 Universal Hashing

Universal hashing [4] is a well established notion in computer science with innumerable applications ranging from the purely theoretical to the very practical. In the latter class,

[6] We note that to implement such a general extractor one needs to consider a *family* of keyed hash functions (not just a single function). When an application needs to extract bits from a given source it selects a random function from the family (by choosing a random key) and applies this function to the given input. Hence, to use such extractors an application needs to have some number of initial random bits available (these bits do not need to be secret and can be re-used to hash multiple inputs from the source). Common applications where this is possible include key-exchange protocols and other key derivation scenarios (see [7]). When such initial randomness is not available, one has to resort to stronger assumptions on the hash function or on the source distribution, or both.

we can mention th use of universal hashing for the construction of the so called Carter-Wegman MAC functions [16]. However, in the context of the design of cryptographic hash functions there has been little interest in this notion. Here we argue that universal hashing is a necessary condition for many of the central properties required from the AHS and hence should have a significant role in the evaluation of hash designs. We start with a definition (which applies to a keyed family).

We say that a family of functions F indexed by a key K is ε-universal if for any two inputs a and b the probability (over the choice of K) that $F(K,a) = F(K,b)$ is at most ε.

Note that this definition is purely combinatorial. Showing that a family is *not* ε-universal means showing a pair of inputs a, b for which more than an ε fraction of hash functions in the family map a and b to the same value. In a computational variant, one requires that it is *infeasible* to find such a pair (even if it may exist). Note that such an attack is against the whole family and not against a specific key. We will use the shorthand ε-AU for ε-universal (the 'A' stands for "almost") and ε-cAU for the computational notion [2].[7]

Here we present several important applications and properties of cryptographic hash functions that require the hash family to be universal. In particular, if one can show that a proposed hash family is *not* universal then one shows that *none* of these properties hold for that family.

We start with PRFs. Let $\{F_K\}_K$ be a family of pseudorandom functions. Now, assume that there is an attacker that can find a pair of inputs a, b for which the probability for random K that $F_K(a) = F_K(b)$ is significant (i.e., noticeably higher than 2^{-n} where n is the output size of F_K). Clearly, this attacker is also a distinguisher that breaks the PRF property (since such collisions are improbable in a random function). In other words, ε-cAU, for small enough ε is a *necessary condition* for a family to be a secure PRF.

Moreover, we note that the possibility of attacking a PRF family via cAU attacks is not just a theoretical possibility. Indeed, recent attacks against HMAC build exactly on breaking the universality property of Merkle-Damgard functions. [5, 12] Roughly, these attacks work by finding differentials Δ such that for a random r, the probability that $H_K(r) = H_K(r + \Delta)$ is high where H_K represents the inner function in HMAC, i.e., the iterated hash function with its IV set to a random value K. Note that it is essential for these attacks not to depend on the actual outputs of H_K since these are not available to the attacker acting against HMAC. The setting of universal hashing captures exactly this attack scenario. Not surprisingly universal hashing also plays a central role in the proof of HMAC; in particular, [2] shows that in the case of Merkle-Damgard schemes, if a compression function is PRF then the iterated construction is cAU. Finally, we point to a more general and fundamental role of cAU functions for the construction of PRFs: cAU provides a general way of transforming fixed input-length PRF into variable input-length PRFs[8].

[7] We note that in a non-uniform model both notions are equivalent even though such a non-uniform attacker may not be "constructive" without having a specific pair that collides for many keys.

[8] That is, if $\{f_K\}$ is a PRF over inputs of length n, and $\{h_J\}$ is a family of cAU functions with variable

Universal hashing is also a necessary condition to achieve collision resistance (for keyed families). Indeed, having a pair of inputs that collides under most functions in the family implies that collisions are easy to find. Moreover, universal hashing is strongly related to the notion of TCR families (that is the reason for naming such families as "universal one-way" as in [13]), a weaker notion of collision resistance sufficient for building secure signature schemes (see Section 2.1).

Finally, we point out to another important property of universal hashing, namely, preserving input entropy. That is, if inputs to a universal family are selected according to a probability distribution with a given min-entropy m (i.e., no element in the distribution has probability higher than 2^{-m}), then the average min-entropy of the output distribution (under a random function in the family) is close to m. Moreover, if the output of the hash family is short enough relative to the input entropy this output is guaranteed to be close to uniform, i.e., universal hashing (with small enough ε) serve as a basis for good extractors [10]. Both (related) properties, entropy preservation and entropy extraction, are central to the uses of hash functions as key derivation functions (see Section 2.4).

In summary, universal hashing provides a well-defined notion that plays a central role in the main properties expected from a good cryptographic hash family, including collision resistance and random-like behavior, and has already been used (implicitly) in recent cryptanalysis work. As such we believe that *universal hashing should be included as an explicit criterion for the design and evaluation of the AHS.*

2.6 On Random Oracles

As we have mentioned already in the introduction being a random oracle, or being "indistinguishable from a random oracle", is not a property that can be achieved by efficient functions, nor it is a property that can be quantified in a well-defined way. The best we can hope for is to find well-defined, achievable, properties of random functions that suffice for specific applications. The preceding sections discuss such properties, and the more of these properties a hash function (or family) has the more confident we can be of the hash design. Yet, not even the combination of all these properties makes a function to "behave like a random oracle". Unfortunately, we still have important cryptographic constructions that cannot be validated without resorting to a random oracle abstraction, including some of the entropy extraction applications (for example, when extracting pseudo-random bits from arbitrary sources using a deterministic unkeyed hash function), or in the well-known "Fiat-Shamir methodology", and many more.

Hence, while we cannot completely avoid the use of random oracles as an abstraction, we have to be very cautious when using actual hash functions in-lieu of these oracles. For example, doing so may easily lead to bad designs. As a trivial example, if we treat a hash function H as a random oracle then the construction $F(K, M) = H(K||M)$ is a perfectly

input-length and n bits of output, then the composed family $f_K(H_J)$, for random and independent J, K, is PRF over variable-length inputs

secure PRF; yet, any instantiation of H with a Merkle-Damgard hash function results in a totally insecure PRF when applied to variable-length inputs (due to extension attacks). Indeed, even if one assumes a compression function to be formed by a family of truly random functions, the resultant Merkle-Damgard hash function is *NOT* a random function. On the other hand, it has been shown [6] that the HMAC scheme built on a family of random compression functions results in a function that is "indifferentiable" from a random oracle in a well defined sense. This does not mean that HMAC, with any real-world compression function, behaves as a random oracle, but at least it prevents structural weaknesses as in the above example. Proposers of new functions and new modes of operation (and iteration) should be encouraged to argue about similar "randomness preservation" properties of their designs.

2.7 Other Functionalities

The preceding sections have covered what we consider are the most central applications and functionalities of hash functions as used in practice. While there are many applications not included in this list, we believe that most of these other applications have requirements that are subsumed by the properties discussed so far.[9] It would be interesting to compile a more exhaustive set of functionalities of hash functions used in practice and check for specific requirements not included here. Examples of applications not explicitly covered here include: one-way transformations for UNIX-style passwords, data fingerprints and digests, hash chains (as in Lamport, S/Key, etc.), Merkle trees, timestamping techniques [8], and commitment schemes. Requirements in these applications range from simple preimage resistance to full collision resistance and, in some cases, may also require (or implicitly assume) some form of pseudo-randomness or entropy extraction[10].

2.8 Modes of Operation

It is one of the central recommendations of this work, that submitters of candidates for AHS specify how to exactly use a proposed function to achieve the core functionalities identified here (e.g., PRF, MAC, KDF). We refer to these (possible) different uses of the hash function as *modes of operation*. The reason that designers of hash functions must provide such modes is that, in general, a plain hash function may not offer all these functionalities. To a minimum, these modes of operation need to include a specification of how to key

[9] Of course, a big exception is the "requirement" to behave as a random oracle which, as discussed in the introduction, cannot be achieved in practice, and therefore not considered as an effective requirement here. Even in this case, the best we can hope for is to have a hash family that achieves as many of the properties discussed earlier.

[10] One interesting example is provided by *perfect one-way functions* [3] where in addition to regular preimage resistance one requires that "no partial information" on the preimage be leaked (this is often required in privacy-sensitive applications, e.g., when "blinding" personal information). Keyed techniques for entropy extraction discussed in Section 2.4 are useful for this application as well.

the function (either with secret keys or non-secret ones). The most common example of a mode of operation for hash functions is HMAC which proposers of Merkle-Damgard functions may adopt. However, for designs that depart considerably from the Merkle-Damgard methodology, a completely new "PRF mode of operation", for example, may need to be defined, including a specific way to key the function.

We note that the security properties required from the basic hash function in order for these modes of operation to securely implement the required functionalities, depend on the proposed hash function and on the proposed mode of operation. Submitters of new functions should specify what are these assumed security properties and how they imply the security of the intended functionality.

An issue raised by the potential need for new modes of operation is how practical it is to replace, say, SHA-1 with a new design that cannot work with HMAC to provide services such as PRF and MAC. The replacement of SHA-1 with another function may be significantly easier than replacing also the HMAC code that uses SHA-1. Indeed, HMAC is many times wired into the specification of standards and applications (and implementations) in a way that makes it hard to replace with a completely new mode of operation. As discussed in the introduction, this is a serious issue that NIST needs to consider. In particular, one may need to contemplate the possibility of deriving HMAC-compatible compression functions from new designs, even if these do not follow the Merkle-Damgard methodology.

Submitters of new functions may also propose "modes of operations" for their design in order to achieve other functionalities, e.g. TCR functions, universal hashing, and other forms of randomized hashing. Some of such modes may also include the option of output truncation as required in some signature applications as discussed in Section 2.1 and which has advantages when using hash functions as extractors [7].

Another interesting contribution may be the specification of "modes of iteration", i.e., different way to iterate a compression function to achieve better properties than, say, Merkle-Damgard iteration. This may include defenses against extension attacks. One such mode could perpend the length of a message to the message before hashing. While this mode will not be applicable in certain cases (when the length is not known a-priori), it will be very beneficial in others, such as in the cases of signing digital certificates, to reduce vulnerabilities due to extension attacks. We note that in such a case, special care needs to be taken by applications not to mix between a mode that prepends length with one that does not, or else attacks may be possible. One way an application (such as the certificate signing one) can solve this problem is by including under the signature an algorithm identifier that ensures that a length-prepending mode was used.

3 Attacks and Thresholds

Here we discuss some of the attack parameters to be considered when evaluating submissions and suggest some threshold values for what is to be considered acceptable security for each of the main functionalities discussed here. This is offered as an input to NIST who may want to do its own security assessment and provide thresholds values.

3.1 Parameters

We identify several parameters involved when quantifying attack complexities. Not all parameters are relevat to all functionalities. We later discuss values for parameters for specific functionalities.

Hash parameters: Output size of hash denoted n.
Key size denoted k (in keyed functionalities).

When talking about input distributions we use m to denote min entropy of the distribution (i.e., each possible input is selected with probability at most 2^{-m})

Attack parameters (assume a fixed computational model):

T = time: For simplicity define time unit = time to compute hash function on fixed size input (e.g. 512 or 1024 bits)

M = memory

Q = queries (in attacks where number of queries make sense, e.g. PRF/MAC)

e = success probability over internal coins

w = success probability over keys (see sub-section 3.3 for more details on relationship with key size k, and other ramifications)

Although the interaction between different parameters may be very complex depending on specific functionalities and attacks, we will simplify this general discussion by decoupling most of these parameters and giving individual thresholds. Finer granularity thresholds maybe considered for specific cases. One exception is that we will keep a combined threshold T/e representing the expected work to reach (a first) success. This presumes a linear increase in probability with time. Cases where this assumptions does not hold may need a different trade-off. For most cases t/e is a good benchmark.

One more parameter not discussed above (and which we will not discuss for individual functionalities) is message/input length. We only consider attacks that work on inputs less than 2^{64} as reasonable. One may want to set some of the other security thresholds to be a function of smaller-size messages (e.g., 512-bit inputs, 1024-bit inputs, 4K-byte inputs, etc.)

3.2 Security thresholds

Here we provide suggestions for possible acceptability thresholds. That is, we will consider an attack "reasonable" only under these suggested complexity thresholds. Infact, we provide a range of acceptable thresholds. NIST may want to do its own assessment, and these ranges serve only as a guide. These are given per functionality (and type of attack). We start with secretly-keyed functionalities, then move on to unkeyed ones, and finally treat non-secretely-keyed, namely salted, functionalities.

In the following, we set as default value for the size of hash output $n = 256$. Security thresholds may be incresed for larger sizes but this may not be necessary (except, maybe, for collisions resistance).

3.2.1 Improving on generic attacks

A generic attack against a functionality is one that works equally well when the function in use is replaced with a random function. *Any* attack that improves on a generic attack needs to be considered as a potentially serious weakness. In some cases, this may be sufficient to disqualify a candidate. In other cases, one may have to judge the potential of that attack being improved and become more practical. But in all cases one has also to judge the reasonability of the attack in practice. Accepting only designs which resist every attack which is better than generic may be too restrictive and may, in particular, make designs too inefficient. Below we give thresholds for which we consider acceptable attack parameters. Of course one still has to judge things on a case-by-case basis, but these thresholds are useful as a general guideline.

One important consideration is the case of candidate hash functions which have a better-than-generic attack "by design". In particular, the cases where the designer has some rationale for the need (such as for improving efficiency) for this "less-than-perfect" security. Submitters should be requested to provide any such rationale with their submissions.

Another issue that requires consideration are attacks that are generic in the sense that they are generic against a "mode of operation". In other words, these attacks may not work as well on random functions, but it is more due to the "mode" (e.g. Merkle-Damgard) rather than the specific function.

3.2.2 PRF

We consider a key size of $k = 256$. It makes sense to expect keys of that size to be available to applications. Even when we settle for a reasonable attack T/e threshold of 2^{128} we still want larger keys to potentially decrease ratio of weak keys (see sub-section 3.3 for more details) or even to cover for some keys (generated, say, by a KDF) with less than 256 entropy.

Attacks: The two main forms of attacks against PRFs are:

1. indistinguishability from a random function with same number of output bits (in the PRF case, output can be of any size less or equal n)
2. Key recovery

For each parameter we give a range of thresholds. An attack with any one of these parameters worse than this threshold should not be considered a real threat.

We start with thresholds for *indistinguishability* for each parameter. The threshold ranges we provide take into account the fact that this is a secret-keyed primitive (see subsection 3.3 for a discussion on this). The numbers are for attacks against a single key.

T/e : Maximal value to be considered practical should be somewhere in the range of 2^{96} to 2^{128}.

Q : Maximal value to be considered practical should be somewhere in the range 2^{40} to 2^{64} (in the PRF functionality, as opposed to MAC, it is very rare to compute the PRF with the same key in 2^{40} points or more)

M : Maximal value to be considered practical should be somewhere in the range 2^{64} to 2^{96}.

w : MINIMAL value to be considered practical should be somewhere in the range 2^{-64} to 2^{-96}.

In the case of *key recovery*, thresholds should be set to the more conservatives in the above thresholds.

3.2.3 MAC

Same considerations should be given here as for PRF, except that indistinguishability attacks are replaced with forgery attacks. This is a harder attack to mount, in general, than indistinguishability, and hence one may go with the threshold for PRF, but on the more liberal end. Another consideration that makes security of MAC less stringent is that attacking a MAC, after the key is not in use anymore, is useless. The only parameter that needs to be more conservative in the MAC case than in the PRF case is the number of queries which in a MAC applications corresponds to the number of messages MACed with the same key, and this can be very large. A value of Q of 2^{64} would however, be more than enough here.

3.2.4 Collision resistance

Recall we assume output size $n = 256$. We consider the popular unkeyed case, namely, where a single function is used to provide collision resistance.

Due to the unkeyed nature, one has to be more conservative, as the work is completely off-line and the gains from a break for an attacker can be huge (as opposed to having to perform a per-key attack in keyed functionalities).

For $n = 256$ we set the generic 2^{128} as the time threshold. Namely, any attack improving on the generic (birthday) attack should be considered very serious. For larger sizes, NIST needs to determine how to scale security. Here we recommend that designs that use larges sizes, say n=512, that explicitly state that there security is less than $2^{n/2}$ (but at least 2^{128}), and provide well-founded rationale for this design choice should be given serious consideration, in spite of the "less-than-perfect" security.

3.2.5 Preimage resistance

One challenge here is to determine the acceptable input distributions to judge preimage resistance on. Uniform distributions, as usually used in theory, is too restrictive, as in practice one may apply the function to very different distributions. Given a specific distribution, we consider a generic attack as one that would work equally well against a random function on that input distribution. More specifically, if the min-entropy of the input distribution is m and the output of the function is n bits then the generic bound (for T/e) would be: $\min\{2^n, 2^m - 1\}$. In most case however, we would recommend calling an attack "reasonable" only if it takes less than 2^{160} complexity, that is we are setting the threshold to $\min\{2^n, 2^m - 1, 2^{160}\}$. We chose 2^{160} in this case since breaking preimage resistance of a function is the most devastating attack one can have (it would break virtually all other functionalities).

For SECOND preimage resistance, one might have the same considerations but reduce the 2^{160} threshold to 2^{128}.

3.2.6 Salted functionalities: TCR as an example

Here we consider thresholds for keyed functionalities, but ones where the key is non-secret. We use the case of TCR functions as an example. Recall the TCR functionality:

Attacker spends time to find $m1$ (precomputation!), then receives key (or salt) K and needs to find $m2$ such that $f_K(m1) = f_K(m2)$.

As discussed in detail in the next sub-section, the effect of a weak-key probability of w, can be tremendously less taxing on the attacker for these non-secret key functionalities, than for secret key fuctionalities. Hence, here we definitely recommend a w value of 2^{-128}, with at most a relazation to 2^{-96}.

3.3 What should be the key size?

Since the minimum key size prescribed for AES is 128 bits, one could argue that a similar key size should suffice for AHS. However, block ciphers have a secret key security requirement, whereas some of the properties required here are non-secret key. It is correctly believed that for the same key size, and similar resistance guarantees, the resources required for an attack can be tremendously smaller for a non-secret key property than for a secret key property. This is best illustrated by the following example, which also helps explain the various probabilities involved.

We will assume a uniform distribution on the k bit keys. Each attack may only work for a fraction of the keys, e.g. by exploiting some porperties of these keys, and we call such a set of keys the *weak-key set* (for the moment, assume that there is only one weak-key set, and we will consider the more general situation later). The attack itself, under the condition that the key was chosen from the weak-key set, may have its own security parameters, i.e. time and memory required, and may also be randomized, and hence have a probability of success associated with it. For example, it may be randomized in the sense that it randomly picks a set of inputs. Thus, an attack can be characterized by three (or four) parameters: (a) the weak-key probability, i.e. the fraction of weak-keys, (b) the success probability of the attack, and (c) time and/or memory required for the attack. For example, once could require that an attack is practical only if the weak-key probability of the attack running in time t, and success probability at least $t/2^{128}$, is at least 2^{-96}. In more formal notation,

$$\Pr_{\text{key}}[\Pr[\text{Time} < t] > t/2^{128}] > 2^{-96}$$

Continuing with the example, we remark that the weak-key probability relates to world wide deployment of applications and their re-keying, whereas the probability over randomness of the attacker relates to the number of independent attackers working on the same key deployment.

So, first consider the (non-secret) keyed target collision resistance (TCR) property, and suppose there is an attack which has a fixed target message P, and for 2^{-48} fraction of the keys, finds a target collision with success probability 2^{-64} in time 2^{32}. Also, assume that an attacker can identify a given key to be weak in a minuscule amount of time. Then a single attacker prompts with P, looking for weak-key deployments, and after 2^{48} such instances succeeds in locating a weak key. Then, in time 2^{32+16} it succeeds in finding a collision with probability 2^{-64+16}. If there are 2^{64-16} collaborators of the attacker, on average one of them will succeed in time 2^{32+16} in finding a collision. Note, that there is no wastage of resources while looking for the weak key.

On the other hand, if the same parameters held for a secret key property, e.g. MAC, then a single attacker must apply time 2^{32} to each re-keying instance, and then succeed with probability 2^{-48-64}. If there are 2^{64-16} attackers, they each must apply time 2^{32+16} and then on average one succeeds on 2^{-48} fraction of re-keying instances. In effect, they must each wait for time $2^{32+16+48}$ before one of them has a hit.

We now demonstrate that if there are multiple weak-key sets, then in the non-secret key situation, the weak-key probability can be further boosted, with other parameters remaining same. This is not the case for secret-key properties, where even though the weak-key probability can be boosted, it comes at a price. So, continuing with the previous example, suppose there are 2^{32} different weak-key sets, each of size $2^k \cdot 2^{-48}$ (and for simplicity, assume that the sets are disjoint). Further, assume that the attacker can quickly identify which weak-key set the given (non-secret) key belongs to. Hence, his effective weak-key probability is now as high as 2^{-48+32}, and hence he needs to prompt only 2^{16} instances before locating a weak-key. On the other hand, in the secret-key situation, even if there are 2^{32} different weak-key sets (e.g. each with its own different linear characteristic), the attacker can boost his weak-key probability only by paying a price, e.g. by keeping statistics for each linear characteristic, and hence essentially requiring 2^{32} memory.

Notice that in the multiple weak-key sets situation, the "weak-key" probability is a misnomer, as what is really meant is the probability (over the choice of keys) that the attack is applicable, as expressed in the formal notation above.

Since, weak-keys are quite possible for non-secret key properties such as collision resistance, it is best to require a large key (e.g. 256 bits), and consider attacks to be practical only if they work on large enough (e.g. 2^{-96}) fraction of keys. Even for secret key properties, e.g. PRF, it is possible that one always finds some weak keys for linear characteristics, especially given that the function is a compression function and not a permutation, and hence it is best to require a larger key but with a threshold on the fraction of weak keys. Further, it helps resolve the tricky situation of having the key size different from the IV size (for traditional compression function designs).

References

[1] Mihir Bellare and Phillip Rogaway, "Collision-Resistant Hashing: Towards Making UOWHFs Practical", CRYPTO 97, LNCS 1294, 1997

[2] M. Bellare, "New Proofs for NMAC and HMAC: Security without Collision-Resistance", CRYPTO 2006.

[3] Ran Canetti, "Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information", CRYPTO 1997.

[4] L. Carter and M. N. Wegman. "Universal Classes of Hash Functions', *JCSS*, 18(2):143–154, April 1979.

[5] Scott Contini and Yiqun Lisa Yin, "Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions", Asiacrypt 2006, pp. 37–53.

[6] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, Prashant Puniya, "Merkle-Damgård Revisited: How to Construct a Hash Function", CRYPTO 2005.

[7] Dodis, Y., Gennaro, R., Håstad, J., Krawczyk H., and Rabin, T., "Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes", CRYPTO 2004.

[8] Stuart Haber and W. Scott Stornetta, "How to Time-Stamp a Digital Document", J. Cryptology 3(2): 99-111 (1991)

[9] Halevi, S., and Krawczyk, H., "Strengthening Digital Signatures via Randomized Hashing", CRYPTO'2006.

[10] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby, "Construction of a Pseudo-random Generator from any One-way Function", *SIAM. J. Computing*, 28(4):1364–1396, 1999.

[11] John Kelsey and Bruce Schneier, "Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work", EUROCRYPT 2005.

[12] Jongsung Kim, Alex Biryukov, Bart Preneel and Seokhie Hong, "On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1", SCN 2006: 242-256

[13] Moni Naor and Moti Yung, "Universal One-Way Hash Functions and their Cryptographic Applications", STOC 1989.

[14] N. Nisam and A. Ta-Shma. "Extracting Randomness: A Survey and New Constructions", *JCSS* 58:148–173, 1999.

[15] Phillip Rogaway, Thomas Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance". FSE 2004, 371-388.

[16] M.N. Wegman and L. Carter. "New Hash Functions and Their Use in Authentication and Set Equality", *JCSS*, 22(3):265–279, July 1981.

[17] Yiqun Lisa Yin, NIST 2nd Hash Workshop, August 2006